Informe del primer sprint

GPS-Fooding

Cristina Lahoz Egea (544393)
Patricia Lázaro Tello (554309)
Jorge Martínez Lascorz (571735)
Alejandro Royo Amondarain (560285)
Jaime Ruiz-Borau Vizárraga (546751)

Índice

1. Introducción

El producto a desarrollar consiste en una aplicación de recetas de comida para smartphones, concretamente dispositivos con sistema operativo Android, que ayude a personas sin grandes conocimientos de cocina a hacer la comida.

El equipo que desarrolla este proyecto se compone de cinco integrantes. Los cinco miembros del equipo componen el *equipo de desarrollo*, Cristina Lahoz es la *dueña del producto* y Alejandro Royo es el *Scrum Master*.

En las siguientes secciones del documento se explica el estado del producto al finalizar el sprint y las diversas estrategias seguidas en el proceso durante el sprint.

En la sección del *Producto* se plantea el estado actual del producto, incluyendo la pila de producto, plan de producto y planificación de lanzamientos, así como su diseño arquitectural.

En la sección del *Proceso* se explican las diversas estrategias de control de versiones, test y construcción automática de software seguidas, así como diagramas de burnup, velocidad del equipo y esfuerzos, y los resultados de la retrospectiva del sprint.

En la última sección, *Conclusiones*, se enumera el grado de cumplimiento de los objetivos de la evaluación del proyecto, así como un pequeño sumario del documento.

2. Producto

2.1. Plan de producto

Nuestra visión

Fooding es una aplicación para Android que conciencia a las personas para que coman sano mediante un sistema de puntuación y rankings globales. Para comer sano no hay nada mejor que el hábito, y Fooding trata de crear precisamente esto, hábitos.

Sabiendo que no todos los usuarios tienen grandes conocimientos de comida, se pone a su disposición una gran colección de recetas con instrucciones detalladas y paso a paso y diversas ayudas, como generación de listas de la compra o búsqueda de supermercados cercanos.

La aplicación está dirigida a estudiantes en pisos de estudiantes y personas jóvenes que viven solas o que no tienen tiempo para comer, pues son las que peor comen, ya sea por dejadez, falta de tiempo o de ganas.

Con Fooding, será muy fácil mantener buenos hábitos de comida, respetando el tiempo mínimo para comer, sin saltarse comidas y sin recurrir a la siempre presente comida basura.

Requisitos de la aplicación

La aplicación permitirá al usuario encontrar las recetas en función de los ingredientes que tenga disponibles en un momento determinado (tres ingredientes como máximo). La búsqueda de dichas recetas se podrá filtrar además mediante nombres, ingredientes a utilizar y por tipo de receta (carne, pescado, verdura, postre y pasta). Si el usuario no tiene algún ingrediente para completar una receta la aplicación buscará el supermercado más cercano y permitirá realizar la compra digitalmente. Para que el usuario pueda mantener un registro de las últimas comidas y repetir fácilmente la aplicación permitirá guardar las recetas favoritas.

Por otro lado, la aplicación fomenta la interacción con otros usuarios utilizando valoraciones sobre recetas y clasificaciones en función de dichas valoraciones. Además el usuario podrá competir con la comunidad compartiendo sus recetas y ganando puntos por ello. Estos puntos permitirán al usuario subir de nivel y ser más visible dentro de la comunidad.

Así, cuando publique alguna receta esta tendrá prioridad respecto al resto de publicaciones. Los puntos se podrán obtener contribuyendo a la comunidad de manera activa, esto es, compartiendo nuevas recetas de nuevos tipos de comida, valorar recetas de otros usuarios o recibir buenas valoraciones de las recetas ya publicadas.

Hoja de ruta

A continuación se describe la hoja de ruta a seguir durante el primer año de desarrollo:

El *primer lanzamiento* corresponde a enero de 2016: incluirá las funcionalidades correspondientes al buscador de recetas, sus diferentes filtros, localización del usuario y supermercados cercanos mediante GPS.

El *segundo lanzamiento* se realizará en abril de 2016: incluirá la primera parte de la comunidad de usuarios, con sistema de puntuación, valoración de recetas, ranking por usuarios y aportaciones por parte de los usuarios.

El *tercer lanzamiento* se producirá en julio de 2016: incluirá la segunda parte de la comunidad de usuarios, con sistema de alianzas y ranking por alianzas.

El *cuarto lanzamiento* corresponde a septiembre de 2016: incluirá la compra automática por internet en supermercados.

2.2. Análisis de riesgos

En cuanto al análisis de riesgos que se llevó a cabo antes de la planificación del primer sprint, podemos decir que, en un principio se identificaron cuatro riesgos:

1. La Base de Datos o el servidor no responden
2. Algún miembro del equipo se ausenta en cualquier reunión.
3. Posibles fallos en la comunicación entre los miembros del equipo.
4. Mala relación entre los miembros del equipo.

De estos cuatro posibles riesgos el más importante y que por tanto, más se ha notado en el desarrollo de este primer sprint ha sido el tres, posibles fallos en la comunicación entre los miembros del equipo, ya que muchas de las reuniones o decisiones tomadas durante ese tiempo se han realizado mediante la aplicación Whatsapp, por lo que, esto ha supuesto un gran problema de perdida de información.

Para solucionar esto se ha decidido utilizar Whatsapp para alguna consulta puntual y para los temas más importantes se realizarán reuniones en las que estarán presentes todos los miembros del equipo. El resto de riesgos presentados anteriormente se han mitigado correctamente con las estrategias de mitigación expuestas en el documento de la fase previa al primer sprint.

2.3. Pila del producto

A continuación se muestran las entradas de la pila que forman la pila del producto en este primer sprint:

Meter recetas (S)
Configuración Inicial (M)
Listado inicial de recetas (M)
Búsqueda por nombre (S)
Búsqueda por tipo (S)
Búsqueda por ingredientes (S)
Registro Usuarios (M)
Login de usuarios (S)
Logout de usuarios (S)
Verificar registro (S)
Login automático (S)

- **Meter recetas:**

  1. Existe una base de datos con la estructura necesaria para almacenar las recetas que mostrará la aplicación.
  2. Dicha base de datos contendrá almacenadas suficientes estructuras para que la aplicación sea utilizable. Dicha medida de "utilizable" vendrá dada por el dueño del producto.

- **Configuración inicial:**

  1. Existe un sistema que permite gestionar todas las solicitudes de la aplicación a la base de datos (No es necesario que esté configurado, esta entrada de la pila se refiere a montar el sistema para su uso/configuración más tarde).

- **Listado inicial de recetas:**

  1. Debe mostrar todas las recetas al usuario en una columna al iniciar la aplicación. La información mostrada para cada receta será una imagen que represente el tipo (verdura, carne, pescado) y el nombre de la receta.

- **Búsqueda por nombre:**

  1. Existe un cuadro de búsqueda donde se puede introducir texto y un botón de búsqueda.

2. Al introducir texto y pulsar el botón de búsqueda, se listen aquellas recetas que contengan ese texto en el nombre.
3. Los resultados obtenidos se mostrarán ordenados de forma ascendente o descendente en función de la elección del usuario.

- **Búsqueda por tipo:**
  1. Existe un cuadro donde se puede elegir el tipo de receta sobre el que se desea filtrar.

  2. Al elegir el tipo, se listen aquellas recetas que sean del susodicho tipo.

- **Búsqueda por ingredientes:**
  1. Existe un cuadro donde se pueden elegir los ingredientes por los que filtrar las recetas.

  2. Al elegir los ingredientes concretos, se listen aquellas recetas que contengan dichos ingredientes.

- **Registro usuarios:**
  1. Existen tres cuadros de texto donde se puedan introducir el nombre, el email y la contraseña del usuario a registrar, además de un botón para finalizar el registro.

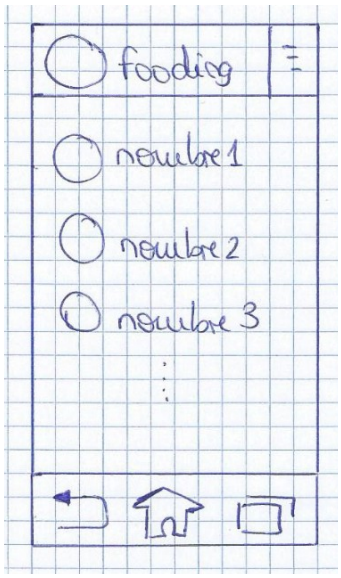
- **Login usuarios:**
  1. Existen dos cuadros de texto donde se puedan introducir el email del usuario y su contraseña, además de un botón de login para validar el inicio de sesión.

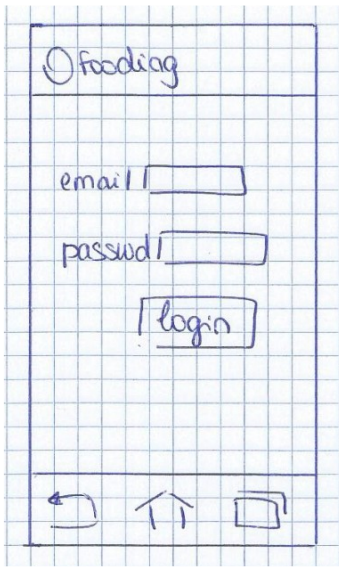
- **Logout usuarios:**
  1. En el menú de inicio cuando haya una sesión abierta de un usuario se mostrará un botón para cerrar dicha sesión y la aplicación volverá al modo inicial de esta.

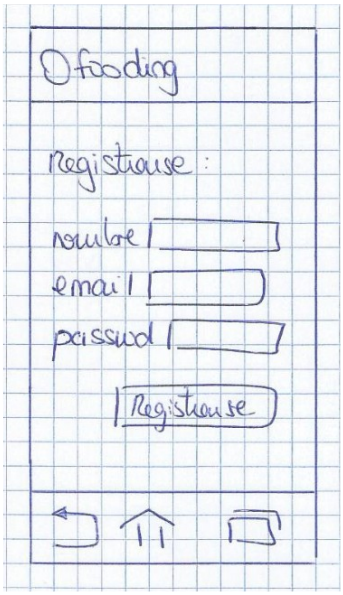
- **Verificar registro:**
  1. Cuando un usuario se registra y pulsa el botón de finalizar se le enviará un correo electrónico a la cuenta de correo indicada con un enlace, en el que tendrá que pulsar para verificar que es la persona que solicita el registro.

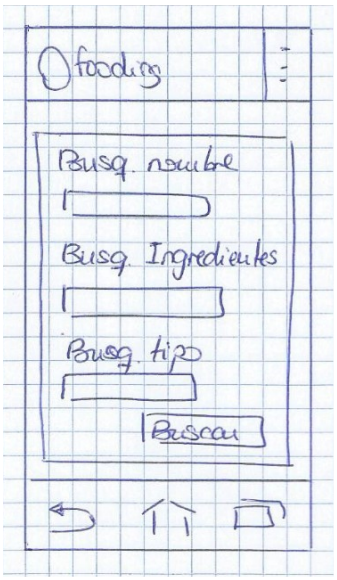
- **Login automático:**
  1. El usuario cuando ya se haya registrado o iniciado sesión anteriormente a no ser que pulse el botón de logout permanecerá con su cuenta iniciada aunque cierre la aplicación.

A continuación se muestran una serie de prototipos iniciales de las pantallas:









Estos son algunos de los primeros diseños de la aplicación.

2.4. Estado de la aplicación

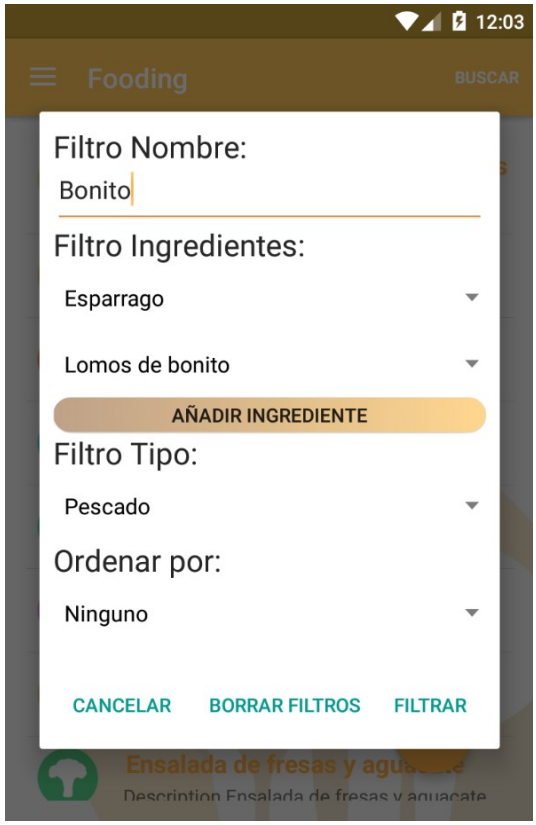
En cuanto al estado de la aplicación, de momento, está desarrollado todo aquello que se propuso el equipo de trabajo para realizar en el primer sprint, las once entradas de la pila ya están implementadas con su correspondiente funcionalidad tanto en la gui de la aplicación como en el servidor y base de datos.

Todas aquellas funcionalidades cumplen la definición de hecho propuesta en la siguiente sección del documento, **proceso.**

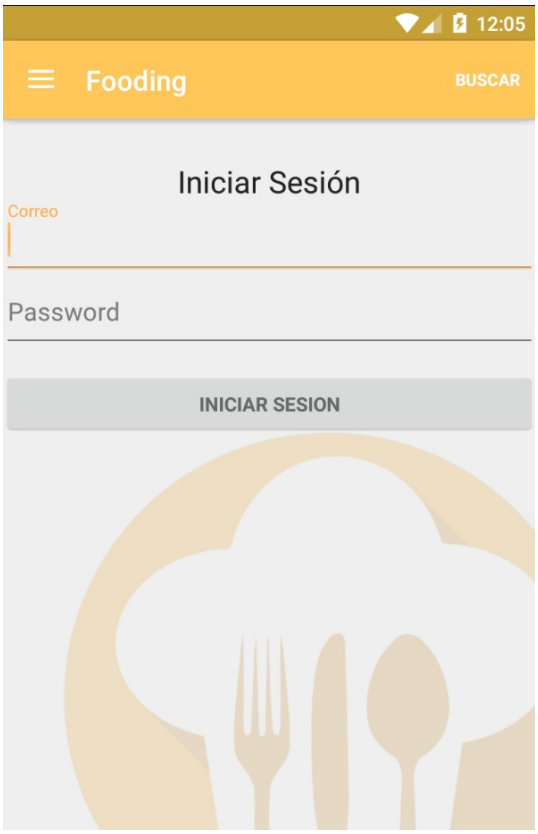
A continuación, se muestran una serie de imágenes con la GUI actual de la aplicación y las funcionalidades desarrolladas.

En cuanto a problemas conocidos se puede destacar la codificación de los caracteres por parte de la base de datos, ya que no reconoce bien las tildes y caracteres especiales, por ello, para el siguiente sprint se ha propuesto solucionar este problema.

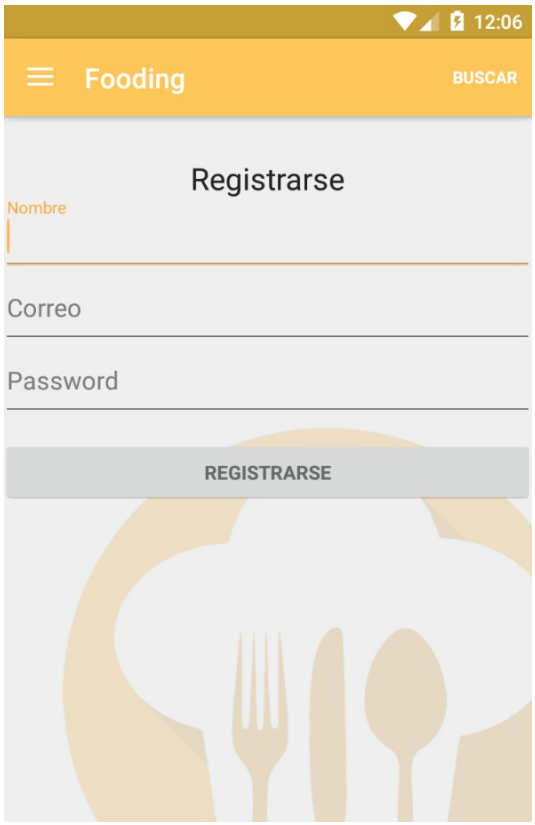
Listado inicial de recetas



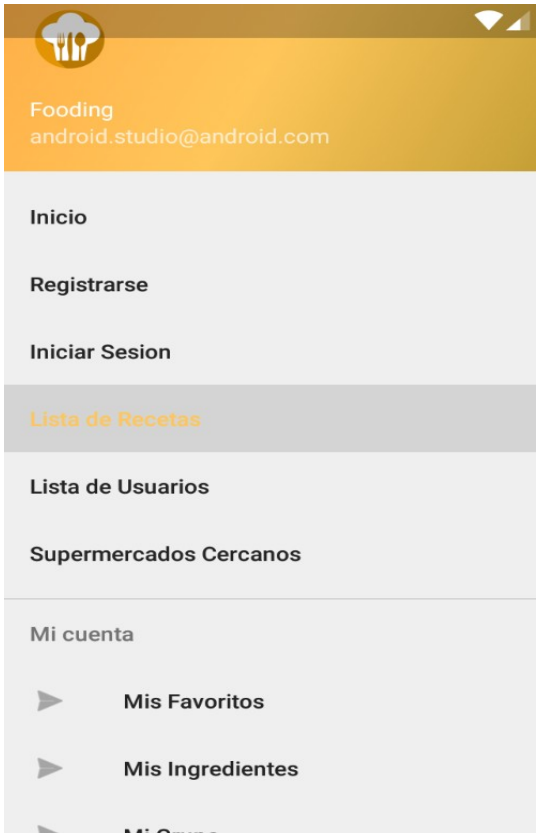
Pop up Búsqueda



Login de usuarios



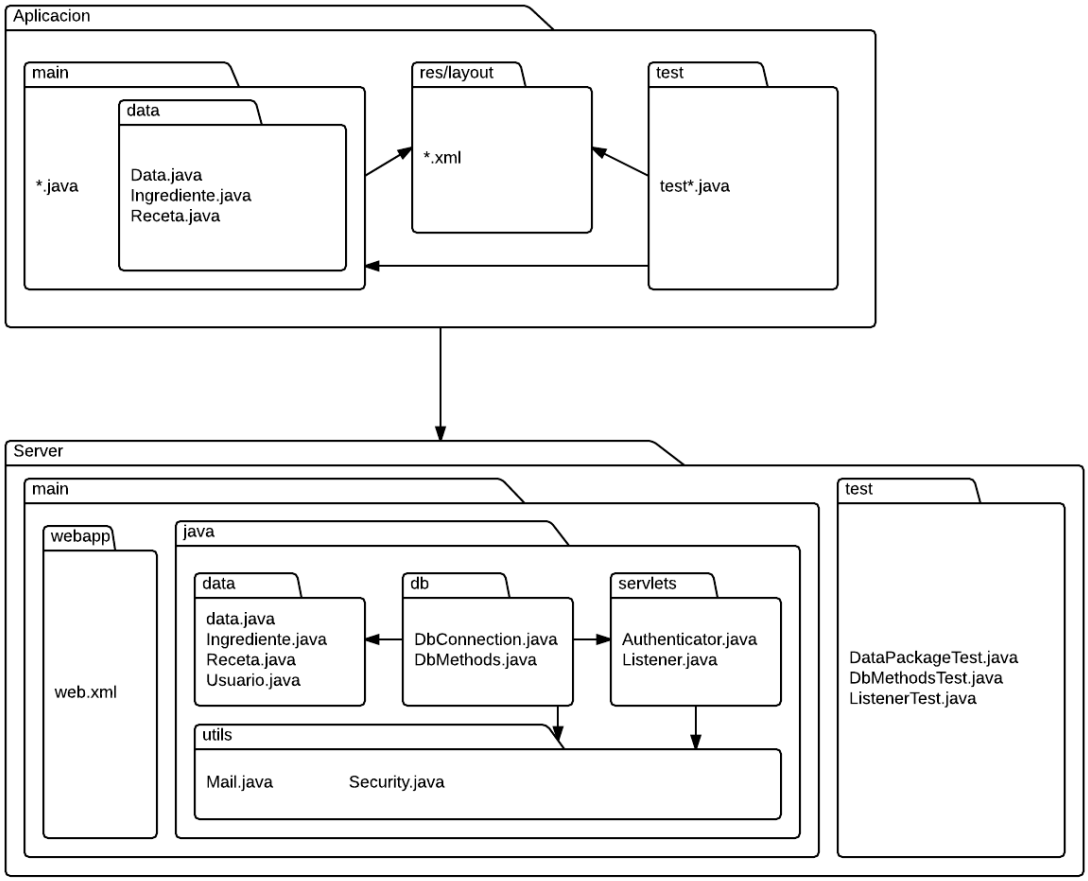
Registro de usuarios



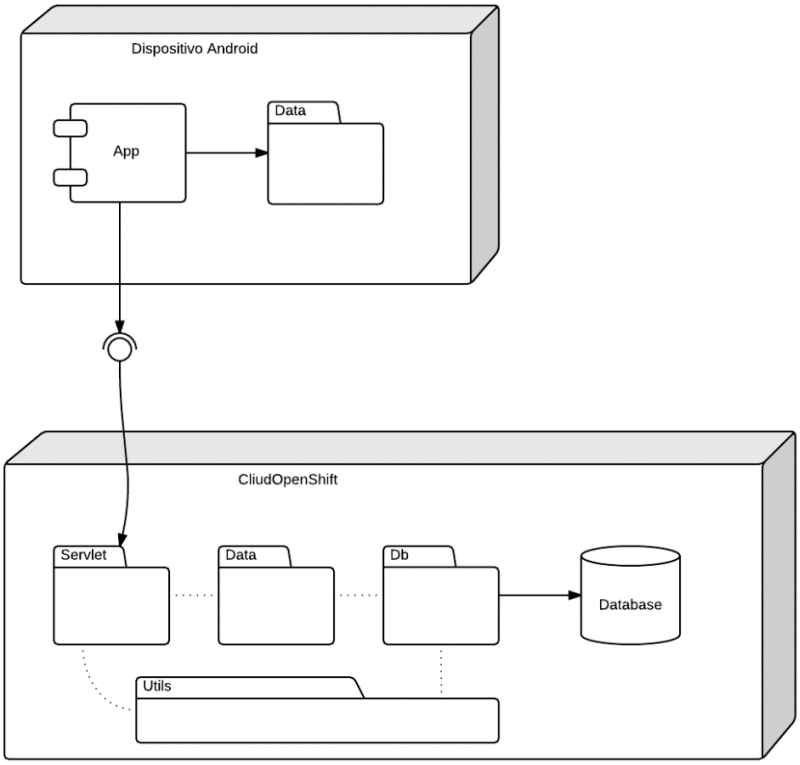
Menú con las acciones

2.4.1. Arquitectura

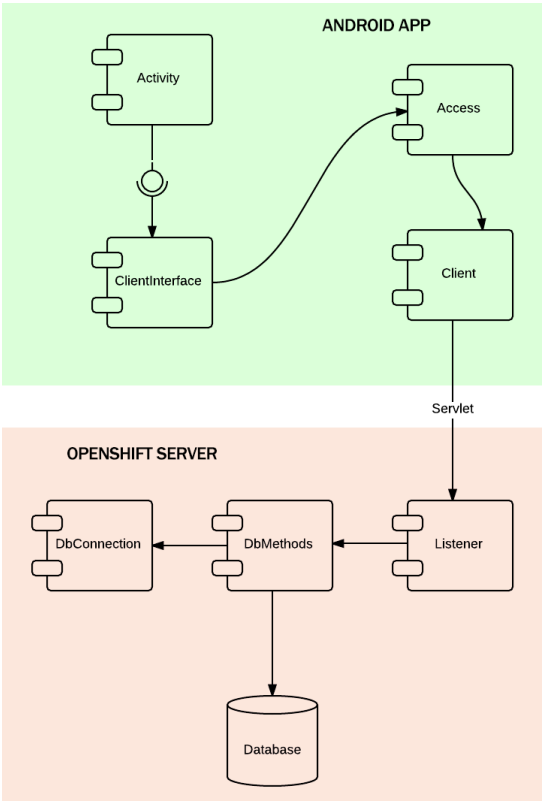
Se ha realizado 3 diagramas de vistas (Paquetes, componentes y despliegue) para diseñar la arquitectura del sistema. A continuación se muestra el diagrama de paquetes con las clases principales:



Principalmente se distinguen 2 grandes sistemas de despliegue, el dispositivo Android y el servidor OpenShift. La aplicación tendrá el papel de cliente, mientras que el servidor se encargará de la lógica junto a la Base de datos.



La conexión entre la aplicación y el servidor se realizará a través de Servlets.



3. Proceso

3.1. Definición de hecho

Se considera que una tarea está hecha cuando, después de las fases de análisis, diseño, implementación y pruebas, sea entregada a la dueña del producto y ésta dé su aprobación.

Concretamente, se sigue el siguiente esquema de checklist:

- El código correspondiente a la tarea se encuentra alojado en los repositorios de GitHub.
- La aplicación de Android y la aplicación servidor han pasado todos los tests automáticos después de incorporar el nuevo código.
- En Openshift (el entorno de desarrollo del servidor) debe estar alojado un WAR con la nueva funcionalidad implementada.
- Se ha notificado a la dueña del producto de la finalización de la tarea.
- El código correspondiente a la tarea se encuentra correctamente documentado; si es necesario, se añadirán nuevas entradas a la Wiki del repositorio, incluyendo los diagramas, imagénes, bocetos... que se consideren necesarios.

3.2. Test y construcción del software

En la aplicación para Android se ha utilizado *Gradle* como herramienta de gestión de dependencias. Para los tests unitarios se ha utilizado la herramienta *Robotium*, pues se ha decidido hacer pruebas de interfaz, probando los métodos y funciones asociadas a las funcionalidades sin llamar directamente a tales métodos.

Para la compilación se ha utilizado también *Gradle*, habiéndose creado un script de compilación que resuelve las dependencias y realiza la compilación de la aplicación, generando un archivo con extensión APK.

No se han utilizado herramientas de cobertura de tests automáticos, pues se ha comprobado al final del sprint el nivel de cobertura de tests manualmente.

En la aplicación servidor se ha utilizado *Maven* como herramienta de gestión de dependencias y compilación. Para los tests unitarios se ha utilizado *Junit* y un mock de *SpringFramework* (para probar los métodos de acceso desde Internet en un servlet, como GET y POST).

La política seguida en el servidor para los tests fue, implementada la funcionalidad, realizar tests en todos los niveles, desde llamadas a los métodos de acceso a la Base de Datos hasta mocks para comprobar el parseo de datos.

Además, se han realizado pruebas adicionales desde la aplicación y desde distintos navegadores para comprobar su corrección.

Para la compilación también se ha usado *Maven*. Los tests son pasados automáticamente con cada push al repositorio del servidor mediante el script de Maven, que también se encarga de crear un archivo WAR y desplegarlo en Openshift.

Se ha utilizado *JaCoCo* como herramienta para la cobertura de tests automáticos, habiéndose implementado un script para Windows para descargar al archivo HTML que genera JaCoCo con cada compilación en el servidor. Adicionalmente, esta herramienta desglosa en paquetes la cobertura de código, además de mostrar el porcentaje de código cubierto globalmente.

3.3. Control de versiones

Se han creado en GitHub un total de tres repositorios: un repositorio para la aplicación, otro repositorio para el servidor y un último repositorio para la documentación.

El repositorio del servidor es una copia exacta del repositorio privado de Openshift en el que se aloja la aplicación servidor: el workflow en este repositorio es *centralizado*, utilizando la rama master como rama principal a la que todos los miembros del equipo suben sus cambios.

Para mantener el repositorio de GitHub sincronizado con el repositorio privado de Openshift se tuvo que crear un enlace a remoto en cada ordenador, así como subir de forma automática al repositorio de GitHub lo mismo que se sube al repositorio de Openshift.

Se decidió crear un conjunto de scripts para automatizar este proceso, siendo estos usados por todos los miembros del equipo. La documentación asociada a estos scripts se encuentra en la Wiki del repositorio, junto a instrucciones de instalación y setup de Openshift.

El repositorio de la aplicación también tiene un *workflow centralizado* en torno a la rama master; aunque para implementar la comunicación entre cliente y servidor se creó una rama (connection_server) separada que iba uniéndose con la rama principal

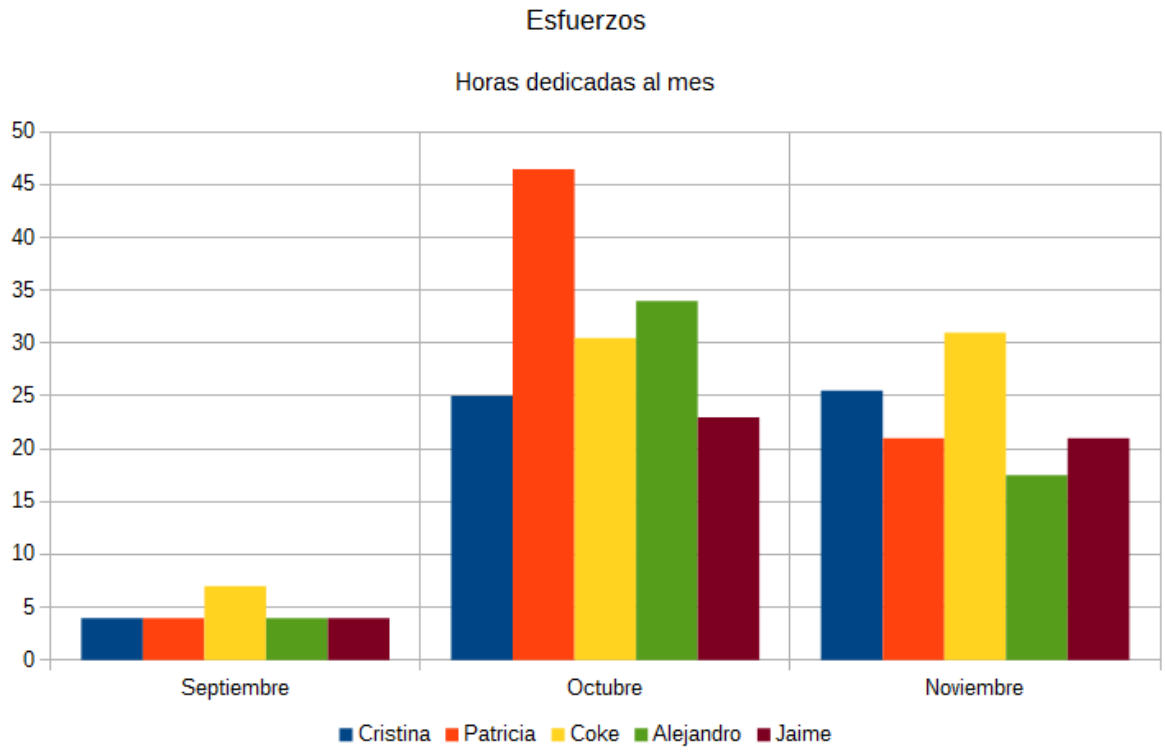
cuando una nueva característica era añadida y testeada.

Por último, en el repositorio para la documentación también se ha seguido un esquema de *workflow centralizado*, pues los cambios en la documentación no son tan frecuentes como para que supusieran conflictos y no se dio por tanto la necesidad de crear ramas adicionales.

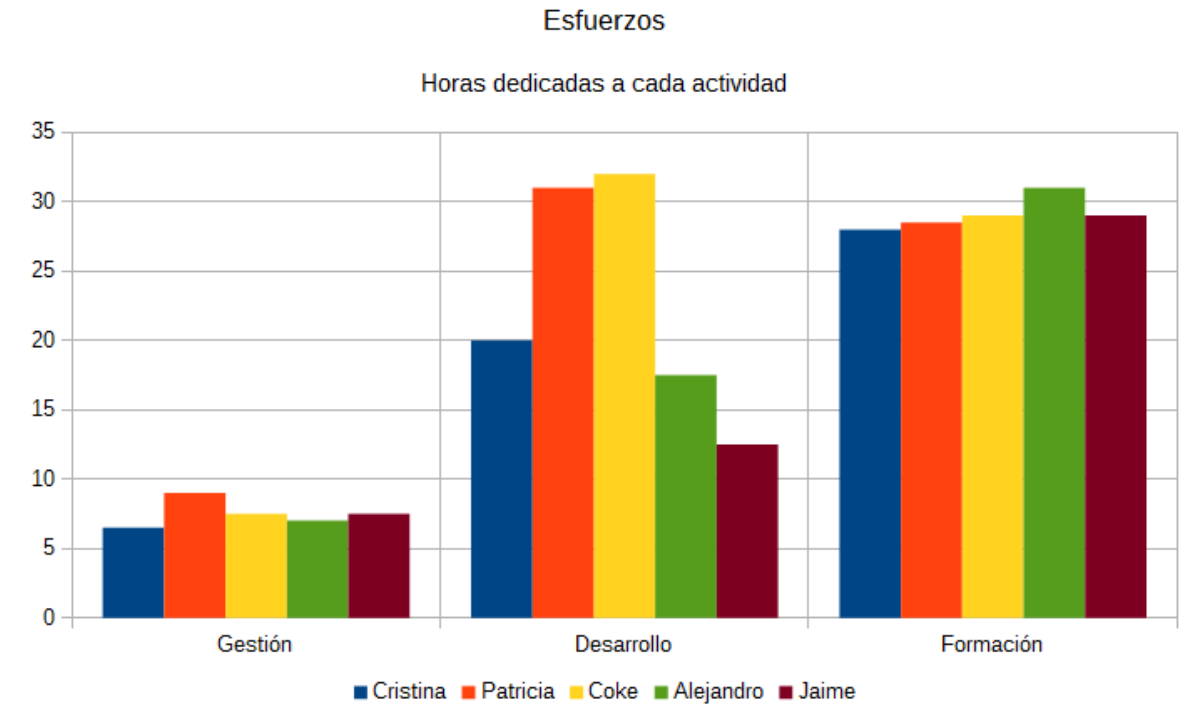
3.4. Estadísticas del equipo

3.4.1. Esfuerzos

Esfuerzos (medidos en horas) de cada miembro del equipo por mes:



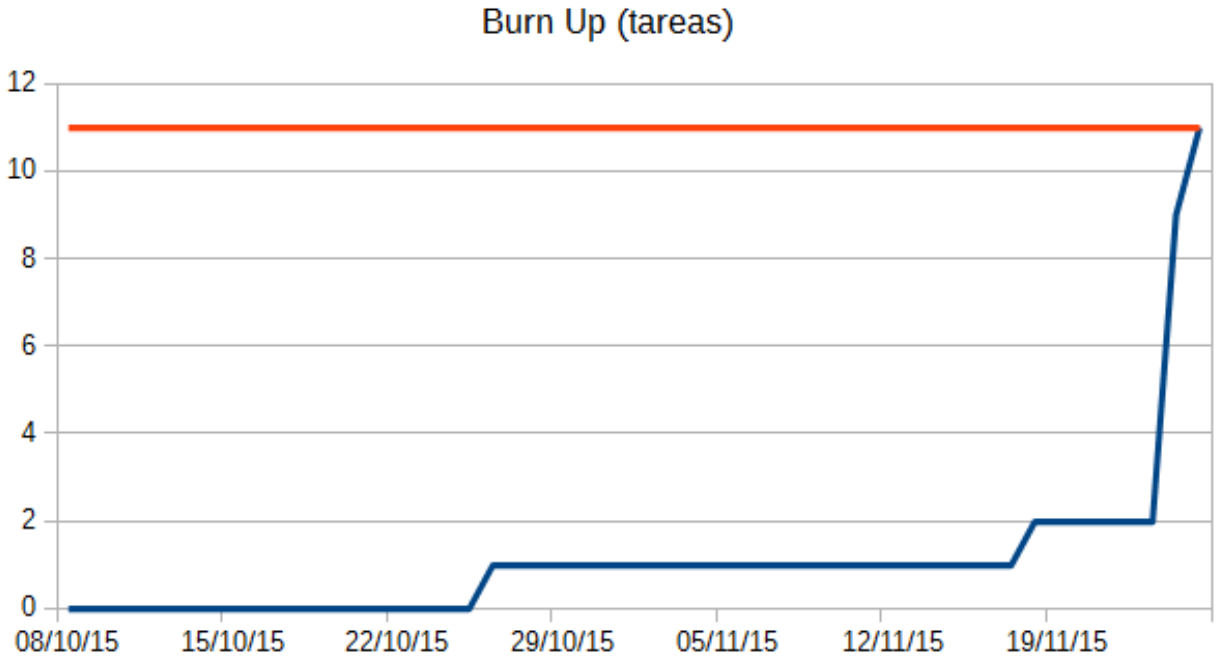
Esfuerzos (medidos en horas) de cada miembro del equipo por actividad:

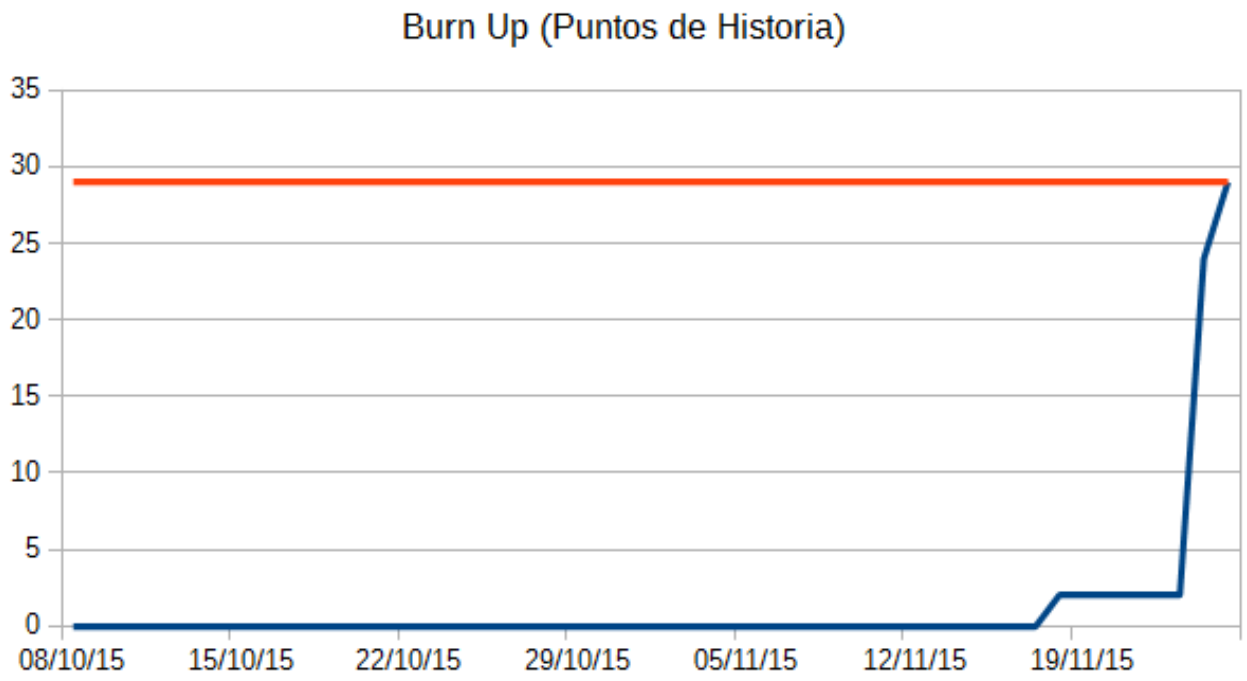


3.4.2. Velocidad

En este sprint, el equipo ha completado 29 puntos de historia; como no se tienen datos históricos del equipo, no se puede dar un rango de velocidades. La velocidad del equipo será, por tanto, *29 puntos de historia*.

A continuación, se muestran dos *diagramas de burn up* o trabajo completado, uno haciendo referencia al número de tareas completadas y el otro con puntos de historia:





La mayoría de las tareas fueron completadas en el último tramo del sprint debido a problemas que impedían pasar los tests automáticos (problemas de configuración).

3.5. Retrospectiva del sprint

Al finalizar el sprint, se ha llevado a cabo la reunión de retrospectiva del mismo, con el objetivo de analizar los problemas surgidos durante el proceso de desarrollo del producto y de dar soluciones o mejoras respecto a dichos problemas.

En la reunión de retrospectiva se sacaron a la luz los siguientes problemas:

- 1) Realización de los tests automáticos de la aplicación. No se han podido automatizar algunos tests, en concreto, los de la GUI.
- 2) Cálculo de la cobertura de código de la aplicación. No se logró configurar para su uso las herramientas para el cálculo de la cobertura de código en la aplicación.
- 3) Uso ineficiente de la herramienta para la gestión del tablero del proyecto. No se ha utilizado Trello adecuadamente para la visualización del estado en cada instante de tiempo de la aplicación, ni se ha mantenido actualizado.

- 4) Falta de una correcta comunicación entre los miembros del equipo. Se ha utilizado Whatsapp para comunicarse en el equipo pero ha quedado patente que se trata de una herramienta ineficiente para la gestión del proyecto, comportando pérdidas de información.

A la luz de los problemas expuestos en los puntos anteriores, se ha discutido la prioridad de solución de dichos problemas y se han realizado las siguientes propuestas para remediarlos (por orden de prioridad):

- En relación con el problema 4, se ha establecido la necesidad de realizar reuniones semanales para poner en común aspectos relevantes de cara al desarrollo del proyecto, forzando al menos una reunión cada dos semanas.

- En relación con el problema 3, ha habido un compromiso por parte de todos los miembros del equipo de corregir su actitud y mantener actualizado el tablero de Trello en todo momento durante el desarrollo de la aplicación.

- En relación con los problemas 1 y 2, se ha optado por dedicar horas del segundo sprint a la solución de estos problemas.

4. Conclusiones

4.1. Sumario

El producto a desarrollar consiste en una aplicación de recetas de comida para smartphones que ayude a estudiantes a hacer la comida.

Inicialmente el equipo se dividió en dos equipos, uno más centrado en la **app,** y otro más centrado en el **servidor,** empleando como medio de comunicación entre ambos "subequipos" **Whatsapp**, y usando Trello como tablero central del proyecto.

Una vez establecida la organización del equipo, se abrieron 3 repositorios para el control de versiones de los ficheros de la aplicación: un repositorio para la documentación, otro para el código de la aplicación móvil y otro para el código de la aplicación del servidor.

Asimismo, durante el desarrollo del proyecto se fueron diseñando en la medida de lo posible tests automáticos que comprobaban el correcto funcionamiento de la aplicación cada vez que se realizaba algún cambio en los repositorios, así como permitir al equipo medir la cobertura del código.

Al finalizar el sprint, a pesar de haber cumplido sobradamente los objetivos que nos propusimos para el mismo, han surgido algunos problemas en relación con la gestión y la comunicación del equipo, pero que se forzará a mejorar y pulir para el próximo sprint.

4.2. Cumplimiento de objetivos

- El código del proyecto se aloja en GitHub y se trabaja de forma habitual contra git: **sí** (mirar los commits y actividad de los integrantes del equipo en los repositorios de GitHub).

- Cobertura de tests automáticos: **77%** en la aplicación servidor y **55%** en la aplicación de Android.

- Historias de usuario/requisitos implementados en el sprint evaluados con buenos criterios de aceptación: **100%** (todos las entradas de la pila hechas han sido validadas por la dueña del producto siguiendo las condiciones de satisfacción planteadas).

- Definición de hecho clara, usada e incluye que se pasen todos los tests automáticos: **sí**.

- Documentación arquitectural (vista de módulos/componente y de despliegue, discusión sobre razones arquitecturales):

- Cumplimiento suficiente de requisitos/características: **sí** (se han terminado todas las entradas de la pila del sprint).

- Control de esfuerzos: **sí** (se sigue el control de esfuerzos propuesto).

- Compilación y gestión de dependencias basada en scripts (despliegue automático, generación de binarios, triggers automáticos...): **sí** (la aplicación servidor incluye despliegue automático, tests automáticos, generación de binarios automática; la aplicación de Android incluye generación de binarios automática).

- Toda la documentación del proyecto está bajo control de versiones: **sí** (la documentación se encuentra en GitHub).

- Tests de aceptación automáticos para criterios de aceptación implementados en el sprint: **6** (búsqueda por nombre, búsqueda por ingredientes, búsqueda por tipo, registro de usuarios, login de usuarios, logout de usuarios).